How Anywhere Computing Just Killed Your Phone-Based Two-Factor Authentication

Radhesh Krishnan Konoth[†], Victor van der Veen[†], and Herbert Bos

[†]Equal contribution joint first authors

VU University Amsterdam, The Netherlands
r.k.konoth@vu.nl,{vvdveen,herbertb}@cs.vu.nl

Abstract. Exponential growth in smartphone usage combined with recent advances in mobile technology is causing a shift in (mobile) app behavior: application vendors no longer restrict their apps to a single platform, but rather add synchronization options that allow users to conveniently switch from mobile to PC or vice versa in order to access their services. This process of integrating apps among multiple platforms essentially removes the gap between them. Current, state of the art, mobile phone-based two-factor authentication (2FA) mechanisms, however, heavily rely on the existence of such separation. They are used in a variety of segments (such as consumer online banking services or enterprise secure remote access) to protect against malware. For example, with 2FA in place, attackers should no longer be able to use their PC-based malware to instantiate fraudulent banking transactions.
In this paper, we analyze the security implications of diminishing gaps between platforms and show that the ongoing integration and desire for increased usability results in violation of key principles for mobile phone 2FA. As a result, we identify a new class of vulnerabilities dubbed *2FA synchronization vulnerabilities*. To support our findings, we present practical attacks against Android and iOS that illustrate how a Man-in-the-Browser attack can be elevated to intercept One-Time Passwords sent to the mobile phone and thus bypass the chain of 2FA mechanisms as used by many financial services.

Keywords: Two-Factor Authentication, Smartphone Security, Financial Trojans, Synchronization, Anywhere Computing

1 Introduction

Approaching an impressive 1.25 billion sales in 2014 with an expected audience of over 1.75 billion, smartphones have become an important factor in many people's day-to-day life [35, 17]. Daily activities performed on these mobile devices include those that can be done on PC as well: accessing e-mail, searching the web, social networking, or listening to music [19]. To enhance usability, both application developers and platform vendors are making an effort to blur boundaries between the two platforms. This is reflected in synchronization features like

Firefox Sync and Samsung SideSync or sophisticated market places like Google Play and Microsoft's Windows Store that allow users to manage their mobile phone remotely.

A second important trend in web computing is the increasing number of applications that provide the possibility to harden user accounts by enabling *2 Factor Authentication* (2FA) for them. 2FA is a form of multi-factor authentication and provides unambiguous identification of users by means of the combination of two different components, i.e., something the user *knows* (PIN code, password) and something the user *possesses* (bank card, USB stick token). With 2FA enabled, if attackers steal a user's password, they still require access to the second component before they can impersonate the victim.

Not surprisingly, software vendors often embody the second component of 2FA in the form of a mobile phone. To authenticate, the web application sends a one-time-valid, dynamic passcode to the user's mobile phone (for instance via SMS, e-mail, or a dedicated application), which must then be entered along with the user's credentials in order to complete the authentication. Since users usually carry their phone all the time, *Mobile Phone 2FA* does not introduce additional costs and can be implemented relatively easy. Examples of well-known companies that provide mobile phone 2FA include Amazon, Apple, Dropbox, Google, Microsoft, Twitter, Yahoo, and many more, including a large number of financial institutions[1]. The latter is represented by many of the biggest financial organisations in the world such as Bank of America, Wells Fargo, JP Morgan Chan, ICBC in China, and ING in The Netherlands.

In this paper, we analyze the security implications of *Anywhere Computing* and show that seamless platform integration comes at the cost of weakening the (commonly perceived) strong mobile phone 2FA mechanism. We define a new class of vulnerabilities dubbed *2FA synchronization vulnerabilities* and show how these can be exploited by an attacker. In particular, we present reliable attacks against both Android and iOS, two platforms that represent a combined market share of over 90% [6]. Our threat model is the same as that of 2FA: we assume that a victim's PC has been compromised, allowing an attacker to perform Man-in-the-Browser (MitB) attacks. In this scenario, mobile phone 2FA should guarantee that the attacker cannot perform authorized operations without having also access to the user's phone. By exploiting certain 2FA synchronization vulnerabilities, however, we show that mobile phone 2FA as used by many online services for secure authentication, including financial institutions, can be easily bypassed.

In more detail, our first attack utilizes Google Play's remote app installation feature to install a specifically crafted *vulnerable* app onto registered Android devices of the victim which is then silently activated and used to hijack One-Time Passwords (OTPs). Our iOS attack, on the other hand, exploits a new OS X feature that enables the synchronization of SMS messages between iPhone and Mac.

[1] `http://twofactorauth.org`

Although the security of 2FA implementations has been subject of prior work [16], we believe that our work is the first to address weaknesses relating to ongoing synchronization and usability enhancement efforts.

Contributions. In summary, our contributions are the following:

1. We identify a new class of vulnerabilities, *2FA synchronization vulnerabilities*, that weaken the security guarantees of mobile phone 2FA.
2. We present practical attacks against Android and iOS that exploit multiple 2FA synchronization vulnerabilities and show how these can be used to successfully bypass mobile phone 2FA.
3. We discuss the security implications of our findings and provide recommendations for various stakeholders. Based on our findings, we conclude that SMS-based 2FA should be considered unsafe.

The remainder of this paper is organized as follows. In Section 2, we outline current efforts deployed by vendors that ease platform integration and provide a definition of *2FA synchronization vulnerabilities*. Section 3 details our attacks against Android and iOS which can be used to bypass mobile phone 2FA. We discuss security implications and recommendations in Section 4, followed by a related work study on the evolution of Man-in-the-Browser attacks and 2FA in Section 5. We conclude in Section 6.

2 Synchronization

To maximize connectivity and to ensure that users never miss another status update, vendors continuously come up with ways to close the gap between PC and mobile devices. In this section, we separate these integration techniques into two categories: (i) remote services as provided by mobile operating system vendors and (ii) integration of applications across the different platforms using synchronization features. Finally, we define *2FA synchronization vulnerabilities* in detail and show example vulnerabilities that we later use to break mobile phone 2FA.

2.1 Remote Services

Mobile operating system market leader Google provides a *remote install* service in its Play Store that allows users to install Android applications on any of their phones or tablets, from a desktop computer. The process is painless and straightforward: a user (i) logs into the Google Play store, (ii) picks an app of his interest, (iii) hits the *install* button, (iv) accepts the app's permissions, (v) chooses the device on which this app should be installed, and (vi) confirms installation. The app is now automatically pushed and installed onto the selected phone—as soon as it has connectivity. Since all the app's permissions are requested and confirmed in the browser already, the only trace left on the phone is a <*app name*>

successfully installed notification message. Similar features have been deployed in app stores of both Microsoft (Windows Phone) and Apple (iOS).

Naturally, platform vendors have adopted security policies to prevent exploitation of this feature. Focussing on Android, for example, Google, deployed two: (i) silent remote install only works for apps on Google Play, which is actively monitored for malware by Google Bouncer; and (ii) newly installed apps default to a *deactivated* state which means that even if the app defines specific event receivers (e.g., on `BOOT_COMPLETED` to start a service at boot-time, or `SMS_RECEIVED` to listen for incoming SMS text messages), it cannot use these until the app is explicitly activated by the user. Activation is triggered by starting the app for a first time, either by selecting it from the launcher or by sending it an intent from another app (e.g., by opening a link from the mobile browser) [1].

In addition to remote install, platform vendors also provide features that help users in locating or wiping a lost device [2, 7, 5].

2.2 App Synchronization

Besides remote *services*, developers try to increase usability even further by incorporating cross-platform synchronization features in their *applications*. This is best illustrated by looking at recent changes in browsers. Browsers once were self-contained software pieces that ran on a single device. Popular browsers like Google Chrome or Mozilla Firefox, however, nowadays offer integrated synchronization services. By using these features, users no longer have to configure browsers individually, but can automatically synchronize all their saved passwords, bookmarks, open tabs, browser history and settings across multiple devices [8, 4]. It is expected that Microsoft's Edge introduces similar functionality soon [32].

Another example of application synchronization is Apple's Continuity which features, among others, synchronization of SMS text messages between iOS (8.1 and up) and Mac OS X (10.10 Yosemite and later): "with Continuity, all the SMS and MMS text messages you send and receive on your iPhone also appear on your Mac, iPad, and iPod touch" [9].

2.3 2FA Synchronization Vulnerabilities

Given the ongoing efforts by both platform vendors and application developers to bridge the gap between the end-user's desktop and his or her mobile devices, we identify a new class of vulnerabilities that, while increasing usability, jeopardize 2FA security guarantees.

Definition *A 2FA synchronization vulnerability is a usability feature that deliberately blurs the boundaries between devices, but, potentially combined with other vulnerabilities, inadvertently weakens the security guarantees of 2FA.*

As an example, consider the previously discussed remote app installation feature: a clear product of a design decision aiming to enhance usability. Although

such option successfully improves usability indeed—users can conveniently manage their mobile device from their browser—it comes with an obvious security risk: if attackers manage to get control over a user's browser, they can extend control to the user's mobile devices as well by pushing arbitrary apps to them. We thus identify the remote install feature as a 2FA synchronization vulnerability.

Focussing again on Android, Google's deployed security measures make that without additional vulnerabilities, attackers cannot abuse this synchronization vulnerability alone to bypass mobile phone 2FA. Finding such vulnerabilities is easy though. First, fundamental weaknesses in Google Bouncer expose multiple ways to bypass malware detection, giving attackers a sufficient time window to push malicious apps to Google Play and thus to mobile devices. Second, we identify numerous ways to activate apps after installation, either by exploiting end-users' curiosity (*hey, what is this app?*) or by relying on additional synchronization vulnerabilities, for example in browser apps: previously discussed features can be used by an attacker to synchronize malicious bookmarks or browser tabs that, when opened on the mobile device, can activate deactive apps.

A second attack exploits the clear 2FA synchronization vulnerability introduced in recent Mac OS X releases. If Continuity is enabled, there is no need for attackers to control a victim's phone: they can read SMS messages from an infected Mac directly.

It is important to realize that 2FA synchronization vulnerabilities are not necessarily caused by bad developer habits or configuration mistakes. More often, they will be the result of a design decision-making process. This means that it is much harder to convince vendors of their mistakes: a 2FA synchronization vulnerability does not leak data or enable code execution, but must be considered within the mobile phone 2FA threat model before it becomes a threat.

3 Exploiting 2FA Synchronization Vulnerabilities

By exploiting the synchronization vulnerabilities discussed in Section 2, we can construct attacks that break mobile phone 2FA. In this section, we present practical implementations of such attacks against the two major mobile operating systems: Google Android and Apple iOS. Additionally, we show that synchronization vulnerabilities also imperil mobile phone 2FA implementations that use a dedicated app to transfer the OTP.

Our attacks operate on the basic threat model of 2FA: we assume that the attacker already has control over the victim's PC, possibly including a MitB, and is specifically interested in bypassing mobile phone 2FA.

3.1 Android

The intention of our Android attack is to exploit the remote install feature of Google Play to push a malicious app onto the user's mobile device. This app can then intercept and forward OTPs sent as SMS messages to a server that is

controlled by the attacker. Given that the attackers have control over the user credentials (stolen by the MitB), this gives them sufficient means to bypass 2FA.

Google's deployed mitigation techniques slightly complicate our scenario. In order to successfully break 2FA, we need to address two defenses: (i) we need to bypass Google Bouncer before we can publish our SMS stealing app in Google Play, and (ii) we need the user to activate the app before it can intercept and forward SMS messages.

Bypassing Google Bouncer Since Google's remote install feature only allows app installation from trusted sources, attackers first need to get an SMS stealing app published in Google Play. For this, they need to bypass Bouncer, Google's automated malware analysis tool that uses both static and dynamic analysis to identify malicious behavior [26]. Once an application is uploaded to Google Play, Bouncer starts analyzing it for known malware, spyware and trojans.

Although the inner workings of Bouncer are kept confidential, prior work has shown that it is easily circumvented [29, 30]. This is confirmed by a recent case study where Avast identified a number of popular Play Store apps that had over a million downloads to be in fact malware [15].

Orthogonal to recent work, our approach to trick Bouncer into accepting rogue apps is publishing a *vulnerable* application [36]. By pushing a poorly coded `WebView` application, for example, attackers no longer have to hide malicious code from Bouncer, but can simply move it to a web server that will be contacted by the app to display regular data [28]. An alternative, even harder to detect scheme, involves exposing a backdoor in native code via a memory corruption vulnerability [11].

To show the practicality of our attack, we successfully published an SMS 'backup' app in Google Play. Upon SMS reception, our app first writes the message content to a file, followed by loading a remote webpage inside a hidden webview component. The prepared webview component, however, is made vulnerable by exposing a `ProcessBuilder` class via the `addJavascriptInterface` API. This allows the remote webpage to execute arbitrary commands within the app's context using JavaScript.

Removing malicious code from the app makes it undetectable for Google Bouncer's static analysis. To also hide from dynamic analysis, we construct the remote webpage in such a way that it does not serve malicious commands when the incoming connection is made from a Google machine. In practice, to avoid accidental misuse, we instructed the webpage to only serve malicious code if accessed from an IP address that is under our control.

App Activation Once installed, Android puts new apps in a *deactivated* state. While deactivated, an app will not run for any reason, except after (i) a manual launch of its main activity via the launcher, or (ii) an explicit intent from another app (e.g., a clicked link from the mobile browser) [22]. Attackers must thus somehow steer their victim into starting the app manually. We identify two reliable approaches to achieve this.

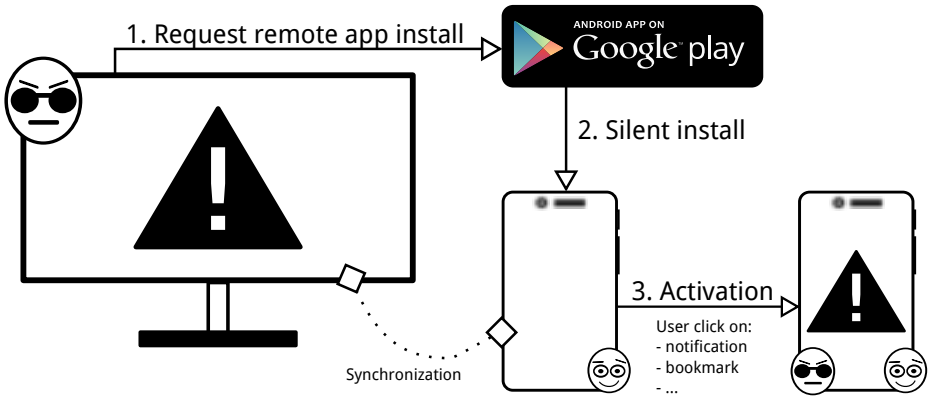

Fig. 1: Malicious app installation process. Attackers (i) use their deployed MitB to request the installation of a vulnerable app, stored in Google Play, and replace all the browser's bookmarks with malicious variants. Google then (ii) pushes the app onto the mobile phone of the victim. Finally (iii) the user is steered into activating the app. Activation is achieved by exploiting browser *synchronization* features to synchronize the malicious bookmarks to the phone, or by exploiting the user's curiosity (a click on the *app is installed* notification message).

1. The most naive method is to hide the malicious activity inside an attractive container. By using a challenging or even provocative app name or icon, a user may be tempted into opening the app manually, simply out of curiosity.
2. Armed with both synchronization vulnerabilities and the victim's Google credentials obtained by the MitB, an attacker can manipulate saved bookmarks, recent tabs, or URLs used in e-mail, cloud documents, social media, etcetera, in such a way that, when clicked, they redirect to a malicious webpage. This page, controlled by the attacker, can then send the aforementioned intent to activate the malicious app.

To prevent a user from detecting the rogue app after it has been activated, we complement it with stealth features. Strictly abiding to the Android developers guidelines, we constructed our app in such a way that, once activated, it removes it's main icon from the launcher. Additionally, we use a name masquerading technique to maximize discretion: (i) the app name shown in the notification bar is different from (ii) the name of the app as found in the launcher, which in its turn differs from (iii) the official app name as shown in the *app overview* (accessible from the settings view). This works because (i) during app submission, the Google Developers Console does not check whether the provided app name matches the official app name as found in the uploaded `.apk`, and (ii) the `<activity-alias>` tag inside the app's manifest allows us to declare additional activity names.

The process of installing a vulnerable app and activating it is shown in Figure 1. The stealthy installation via bookmarks (or recent tabs or some other object of synchronization) combined with name obfuscation makes it hard to tell that an app is malicious, even for experienced users.

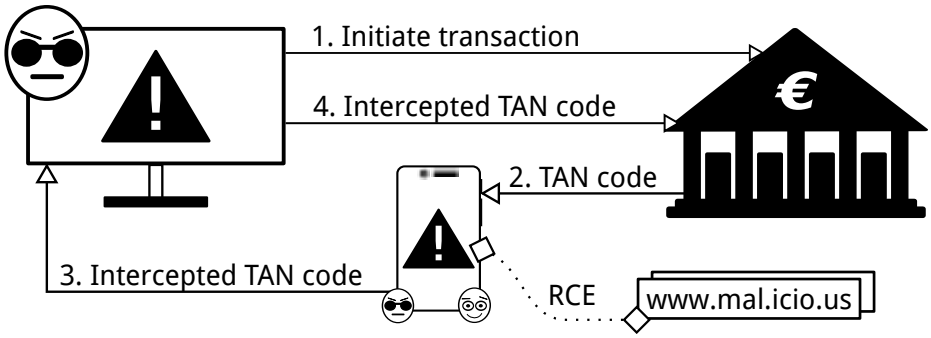

Fig. 2: Completing fraudulent transactions while bypassing 2FA. After our app processes the TAN code, it loads a remote webpage into a `WebView` component that allows the attacker to perform Remote Code Execution (RCE). This way, attackers can hide their malicious activity from Google Play.

Breaking 2FA With the malicious/vulnerable app and activation methods in place, attackers can start their attack from the hijacked browser by requesting remote installation for the rogue app. We implemented a MitB trojan for the Google Chrome browser that can do this. Once installed, our extension can use Google session cookies to start remote app installation and prepare app activation. The plugin basically consists of three phases:

1. **Hijack a Google session**. Our plugin waits for a Google authentication cookie to become available. This happens when the user logs into a Google component (e.g., Gmail, YouTube, Drive, etcetera). Optionally, it forwards the typed credentials or cookies over the network to the attacker.
2. **Remote install**. Using the hijacked Google session, the trojan sends a request to Google Play to retrieve a list of *Device ID*s of all Android devices linked to this particular Google account. Next, for each device, the plugin requests remote installation of the vulnerable app. Since app permissions are approved from within the PC-based browser only, the app will be silently installed, leaving only a <*app name*> *successfully installed* installation notification on the device.
3. **Activation**. In order to allow app activation, our extension rewrites all stored bookmarks and recent tabs so that they point to an attacker-controlled page while the original URL is provided as parameter: `http:\\mal.icio.us\proxy.php?url=<original_url>`. When opened using the mobile Chrome browser, this page performs a redirect to `rogueapp://<original_url>` which triggers activation of the rogue app. The app then immediately fires another intent that redirects the mobile browser to `<orignal_url>`, leaving practically no footprint.

Once activated, the malicious app can be used in conjunction with the PC-based trojan to successfully bypass mobile phone 2FA. Fraudulent financial transactions, for example, can be initiated by attackers once their PC-based trojan has captured banking credentials of their victims. To confirm such transaction, the mobile component intercepts the OTP sent via SMS, and forwards it to the attacker. This attack scenario is depicted in Figure 2.

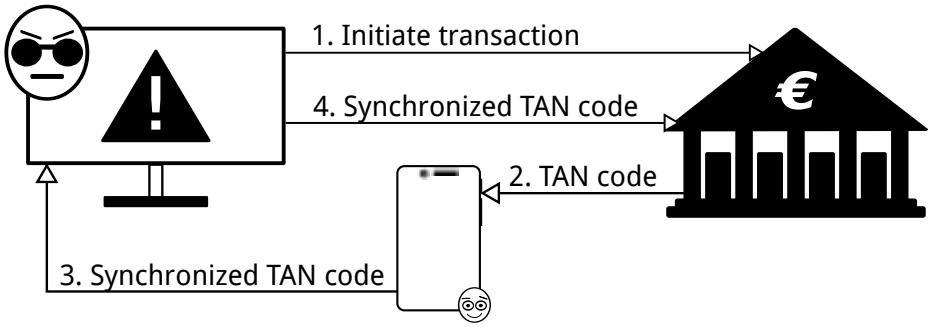

Fig. 3: Breaking 2FA on Apple Continuity. If enabled, Mac OS X 10.10 automatically synchronizes SMS messages between different Apple devices, breaking the second factor.

3.2 iOS

Similar to our Android attack, mobile phone 2FA on the iOS platform can be bypassed by publishing a rogue app to Apple's App Store and installing it from an infected PC via the iTunes remote-install feature. Wang et. al., already demonstrated how a vulnerable app could slip through Apple's strict review process and how such app can be used to access private APIs reserved for system apps to read SMS messages [36, 3]. Additionally, Bosman and Bos showed how a vulnerable app and *sigreturn oriented programming* allow to execute any set of system calls needed to pull of any attack [11].

As of iOS 8.3, released in April 2015, however, it is no longer possible to receive a so-called `kCTMessageReceivedNotification` to let an app act on incoming text messages without using a specific entitlement (similar to the Android `RECEIVE_SMS` permission). Since this functionality stems from a so-called private API, requesting such permission violates the App Store Review Guidelines and will result in an app rejection, effectively breaking this type of attack. The recent release of Mac Os X 10.10 Yosemite, however, opens up a new attack scenario.

As outlined in Section 2, Mac OS X Continuity features options to synchronize SMS and MMS text messages between multiple Apple devices. When enabled, SMS messages that are received on a linked iPhone, are forwarded and stored in plain-text in the `~/Library/Messages/chat.db` file on the Mac.

Breaking 2FA With Continuity enabled, attackers can break 2FA by instructing their MitB to monitor the `chat.db` database for changes and forward new messages to a remote server immediately after receipt. To show the practicality of this attack, we implemented a Firefox extension that uses the `FileUtils.jsm` API to read contents of synchronized SMS messages as soon as they are delivered to the iPhone.

The Continuity attack is illustrated in Figure 3.

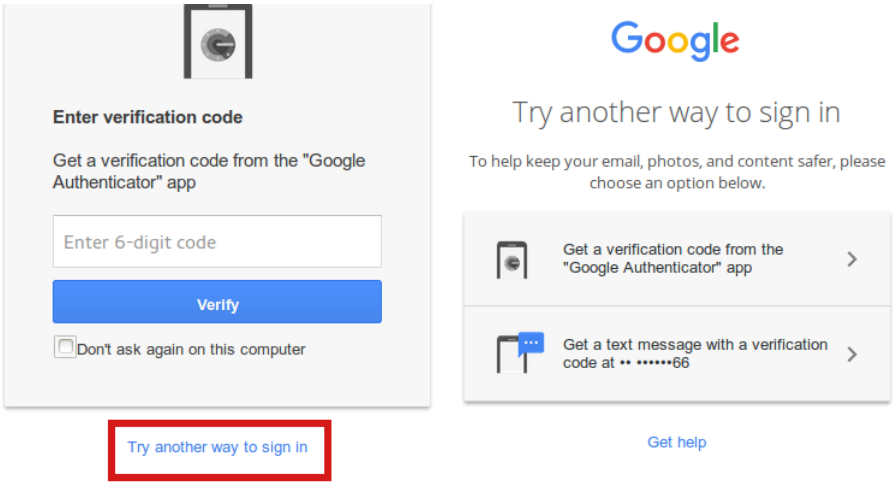

Fig. 4: Bypassing dedicated 2FA apps. The screenshot on the left shows Google 2SV requesting a verification code from the Google Authenticator. Note the *Try another way to sign in* option near the bottom of the window. When clicked, the right-hand figure shows the fallback option to get a text message with an OTP sent over SMS. An attacker in control of the PC-browser is therefore able to dicate what 2FA technique is used.

3.3 Dedicated 2FA Apps

Many online and offline applications are in the process of complementing their authentication mechanism with an optional 2FA step, often dubbed Two-Step Verification (2SV). Open source implementations are provided by Google (Google Authenticator) and Microsoft (Azure Authenticator) and can already be enabled for dozens of popular services, including Google, Microsoft Online, Amazon Web Services, Dropbox, Facebook, WordPress, Joomla, and KeePass.

Due to sandboxing techniques, our previously described attacks cannot access OTPs that are generated by 2SV authenticator apps. During the process of setting up an authenticator app, however, users are advised to provide the underlying system a backup phone number. The rationale behind this is that if, for some reason, users fail to access the authenticator app, they can fallback to requesting an OTP sent over SMS.

Assuming that many users provide a backup phone number that is used by the same smartphone that runs the authenticator app, an attacker can easily bypass these dedicated 2FA apps: (i) having access to stolen credentials harvested by the MitB, an attacker initiates the login procedure; (ii) for logins via the Google Authenticator, for example, when prompted to enter a verification code, the attacker instructs the login page to *try another way to sign in*, followed by selecting the *Send a text message to your phone* option. From here, our previously described attacks can be used to completely bypass the 2FA mechanism.

Figure 4 illustrates how an attacker can fallback to SMS based OTPs when using Google Authenticator.

4 Discussion

In the previous sections, we showed how an attacker can bypass a variety of mobile phone 2FA mechanisms by exploiting synchronization vulnerabilities. We now study feasibility and practicalities of our attacks in more detail. Additionally, we discuss our efforts regarding responsible disclosure, as well as recommendations for involved parties.

4.1 Feasibility

Reviewing our Android attack described in Section 3.1, we conclude that exploiting synchronization vulnerabilities to bypass 2FA can be done in a reliable and stealthy way on Google's mobile operating system. Attackers can reduce their footprint to a bare minimum by breaking the attack down in different steps: (i) a **preparation phase** wherein attackers acquire access to infected PCs, possibly via a *Malware as a Service*-provider [14]; (ii) an **app-installation phase** wherein attackers push a vulnerable app to Google Play and instruct their victims to remotely install it. Depending on the target audience of the attacker, this can be done within a time window of only a couple of hours, after which the rogue app can again be removed from Google's servers; (iii) an **app-activation phase** wherein attackers gracefully wait until victims activate the malicious app. Our app-hiding tricks make that attackers can safely wait days so that a large group of victims get to activate the rogue app; and (iv) an **attack phase** wherein attackers perform an automated attack that requires access to OTPs sent over SMS. One typical example of such attack is transferring funds from saving accounts to an account that is controlled by the attackers.

Although more prerequisites must be met for our iOS attacks described in Section 3.2, they complement each other nicely: the *vulnerable app* approach does not work on iPhones running the latest iOS version, while our Continuity attack requires that victims *do* use more up to date versions of iOS and Mac OS X. The latter, however, also requires that (i) victims have enabled message synchronization (which setup process requires interaction with both Mac and iPhone), and (ii) both devices are connected to the same wireless network. Although this does not necessarily make the attack less feasible, it may slightly reduce its scalability given that synchronization is off by default and increase the detection rate by attentive users (the content of received SMS messages will pop up on both devices).

Finally, although the remote-install 2FA synchronization vulnerability is also prevalent on the Windows Phone (WP) platform, Microsoft does not (yet) provide an API for reading received SMS messages programmatically. Additionally, to the best of our knowledge, WP does not provide SMS synchronization features like Apple's Continuity. It is because of this that we were unable to break mobile phone 2FA on WP.

4.2 Recommendations and Future Work

An important step towards preventing the presented sophisticated MitB-based attacks against mobile phone 2FA, is to raise awareness among the various stakeholders. *Mobile platform vendors* should be aware that the release of new synchronization features may introduce security risks for their end-users. As such, vendors should be extremely careful when enabling new features by default instead of making them optional. It is their obligation to inform *end-users* that enabling or using certain synchronization features might jeopardize security guarantees of mobile phone 2FA. Only then can the user make a considered decision to give up security in favor of usability.

Reviewing our proposed attacks, this means that Apple, for example, should warn users about potential security risks when they set up Continuity. Moreover, if the user decides to enable this feature, synchronizing only messages sent by *trusted* phone numbers — those that are found in the user's contact list — would eliminate our attack scenario, assuming that TAN codes are sent by an unknown sender or SMS gateway. Additionally, we recognize a major task for platform vendors to safeguard their remote-install features. In our view, users should always be forced to explicitly approve new app installations on their mobile device. This way, attackers can no longer silently push apps, but always require manual user-interaction. Ignorant users may still be phished into approving unknown install requests, of course, but such change would eradicate our completely automated attack scenario. We believe that the current app-activation security policy alone as deployed by vendors is too weak, given that additional synchronization vulnerabilities can be used to achieve activation.

Startled users who do not want to wait for a fix from their vendor, can protect themselves from exploitation by using a separate account for each device. This way, remote-install features have zero knowledge about which devices an app can be pushed to. Naturally, the downside of such approach is losing the ability to use synchronization features at all. Authenticator users, in addition, should update their settings so that their backup is a phone number that is attached to a dumb phone. These phones are remarkably harder to get infected.

Besides raising user-awareness, future work should focus on the detection of SMS stealing apps at runtime, given that existing mobile Anti-Virus apps are useless to this respect—they are confined to their own filesystem sandbox and thus cannot access directories of other apps, monitor the phone's file system, or analyze dynamic behavior of installed applications [31]. Instead, system modifications that can monitor the global smartphone state are required. To this, the redesigned permission model of Android Marshmallow in which apps are no longer automatically granted all of their specified permissions at install time, but rather prompt users to grant individual permissions at runtime, is promising. Unfortunately, this model will only be used by applications that are specifically compiled for Marshmallow and can thus still be bypassed.

As an ultimate resort, we recommend that financial institutions consider the removal of mobile 2FA from their business processes and switch to token based 2FA instead—such token must of course be able to show transaction details, so

that Man-in-the-Middle attacks can be detected by the user during transaction processing. Naturally, such switch will cause large expenses; each institution will have to consider whether moving away from mobile 2FA is feasible by comparing costs, gained security, and risk analysis results. Even so, given the attack scenarios we conclude that 2FA on smartphones is currently entirely compromised and no safer than single factor authentication.

4.3 Responsible Disclosure

To show the practicality of bypassing Google Bouncer, we uploaded a first version of our SMS stealing app to Google Play on July 8, 2015, where it has been publicly available for over two months. The app got removed on September 10, 2015, only a few hours *after* we had shared its name and a video demonstration of our attack with the head of Android Platform Security, while we already reported our attack scenario and recommendations to the Android security team months before the initial publication. Responses so far, unfortunately, indicate that Google believes that our proposed attack is not feasible in practice, despite all evidence to the contrary (including actual demos[2]).

We notified Apple about our findings on November 30, 2015, but we did not receive a technical response.

5 Background and Related Work

In this section, we provide a brief historical overview and related work discussion of the two fundamental components covered in this paper: Man-in-the-Browser attacks and Two-Factor Authentication. Additionally, we discuss current, state-of-the-art attacks against mobile-phone 2FA which rely on cross-platform infection. We focus on online banking schemes in particular, as this always was, and still is, one of the services subject to a vast amount of criminal activity.

5.1 Man-in-the-Browser

At first, online financial services depended completely on single-factor authentication (e.g., by using a secret key). For attackers, keyloggers were enough to steal credentials of associated users. However, they also generated vast amount of useless data, forcing the attacker to parse a huge amount of log output in order to retrieve meaningful credentials. Parsing keylog data was considered a challenging and time consuming task for an attacker, as it is hard to automate. As an alternative, cyber criminals deployed phishing campaigns, followed quickly by form grabbing attacks. The latter proved to be an effective and robust mechanism to steal useful information.

Well known banking trojans like Zeus and SpyEye were the first to implement form grabbing by hooking web browser APIs [24, 38]. The fundamental idea

[2] `https://youtu.be/k1v_rQgS0d8`

behind form grabbing is to intercept all form information before it is sent to the network via HTTP requests. Form grabbing can be implemented in different ways: (i) *sniffing* all outgoing requests using a PCAP-based library—something that has the disadvantage of only working for unencrypted data [34]; (ii) *API hooking* the browser's dynamic library to steal all the requests and responses made by the user before they get encrypted [34]; and (iii) using a *malicious plugin* to easily register callbacks within the browser for events like *page load* or *file download* in order to intercept any request or response.

Malicious plugins and API hooking techniques can be used to do more than just form grabbing. Using a plugin, an attacker can modify HTTP responses received by the browser or covertly perform illegitimate operations on behalf of the user. This is commonly known as a Man-in-the-Browser (MitB) attack [21].

Guhring has identified various ways of which a trojan can perform a MitB attack and discusses pros and cons of various countermeasures that could be taken [21]. Boutin studies how webinjects are used by a trojan in the browser and discusses the underground economy behind selling webinjects [12]. Buescher et al., analyzed different types of hooking methods as used by financial trojans [13]. They propose an approach for detecting and classifying trojans by looking at the manipulations they perform on a browser. However, their approach is mainly based on detecting API hooks. As a consequence, MitB attacks that are implemented using plugins cannot be detected using this technique.

5.2 Two-Factor Authentication

Most account fraud and identity theft relate to accounts that use only single-factor authentication [20]. To defend against MitB attacks, financial services started using different types of multi-factor authentication mechanisms. The most elementary mechanism is that of a list of Transaction Authorization Numbers (TAN codes) as provided by the online service, from which the user can choose one to perform a secure transaction. A more convenient method that has been adopted by a majority of financial services is generating a new TAN code for each transaction and sending this via an out-of-band channel to the user. Naturally, SMS is a cheap and efficient candidate channel: almost everybody owns a mobile phone.

To defend against MitB attacks that hijack an ongoing transaction by modifying its details (receiver's bank account number or the amount of money transferred), financial services are starting to include transaction details along with the TAN code in the out-of-band SMS message. Users can then verify the transaction by inspecting these details in the SMS and only confirm if these match their expectation.

On August 8, 2001, the Federal Financial Institutions Examination Council agencies (FFIEC) issued guidance entitled *Authentication in an Electronic Banking Environment* [20]. FFIEC encourages financial institutions to use mobile phone-based 2FA as described above to secure their user's transactions.

Aloul et al., show how an app on a trusted mobile device can be used for generating one-time passwords, or how a mobile device itself can be used as a

medium for out-of-band communication to financial services [10]. This is what most current deployed 2FA implementations use today. Mulliner analyzes attacks that target SMS interception in general and shows how a smartphone trojan can steal OTPs received via SMS. He proposes to use a dedicated channel which cannot be controlled by normal applications for receiving the OTP [27]. This is based on the assumption that mobile trojans do not have root privileges. Schartner et al., describe an attack against SMS based OTPs in the scenario where a transaction is made from the mobile device itself [33]. Since the transaction involves a single device (smartphone), a malware in the device can sniff both credentials and OTPs received via SMS.

Konoth et al., describe how Google's 2FA implementation can be bypassed using a MitB attack on an untrusted device [25]. Dmitrienko et al., analysed 2FA implementations of major online service providers such as Google, Twitter, Dropbox and Facebook [16]. Their work identifies various weaknesses in existing implementations that allow an attacker to bypass 2FA and also illustrates a general attack against 2FA. However, unlike ours, their attack relies on complex cross-platform infection.

5.3 Cross-platform infection

Cardtrap.A is the first discovered malware that features a cross-platform infection implementation. The trojan first infects a symbian smartphone. When the user inserts the memory card of the mobile phone into a Windows PC, it attempts to infect the PC [23]. In 2006, researchers found that it is possible for PC malware to infect a smartphone by exploiting Microsoft's ActiveSync synchronization software [18]. Furthermore, Wang et al., explain how a sophisticated adversary can spread malware to another device through a USB connection [37]. Finally, Dmitrienko et al., demonstrated via prototypes the feasibility of both PC-to-mobile and mobile-to-PC cross platform attacks [16].

6 Conclusion

With the ongoing integration of platforms—the result of a strong desire for enhanced usability—keeping our web accounts safe has become increasingly challenging. In this paper, we showed how synchronization features and cross-platform services can be used to elevate a regular PC-based Man-in-the-Browser to an accompanying Man-in-the-Mobile threat which can be used to successfully bypass mobile phone 2FA. The root cause is that imprudent synchronization functionality has obliterated the security boundaries on which 2FA solutions depend.

Due to the large number of financial institutions that rely on mobile phone 2FA for secure transaction processing, we expect that cyber criminals extend their activities by implementing attacks similar to ours, putting those institutions and their customers at risk. We hope that this paper helps in identifying issues with respect to cross-platform integration and that both software and

platform vendors adopt our recommendations in order to prevent these types of attacks from becoming a major threat in the near future.

Acknowledgements

We would like to thank the anonymous reviewers for their valueable comments and input to improve the paper. This work was supported by the MALPAY project and by the Netherlands Organisation for Scientific Research through grants NWO 639.023.309 VICI "Dowsing" and NWO CSI-DHS 628.001.021.

References

1. Android intents with chrome. https://developer.chrome.com/multidevice/android/intents
2. Find a lost phone. http://www.windowsphone.com/en-us/how-to/wp8/settings-and-personalization/find-a-lost-phone
3. Get SMS broadcast with text body without Jailbreak BUT private frameworks in IOS. http://stackoverflow.com/questions/26642770/get-sms-broadcast-with-text-body-without-jailbreak-but-private-frameworks-in-ios
4. How do I set up Sync on my computer? http://support.mozilla.org/kb/how-do-i-set-sync-my-computer
5. iCloud: Erase your device. https://support.apple.com/kb/PH2701
6. Mobile/tablet operating system market share. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1
7. Remotely ring, lock or erase a lost device. https://support.google.com/accounts/answer/6160500
8. Sync tabs across devices. http://support.google.com/chrome/answer/2591582
9. Use Continuity to connect your iPhone, iPad, iPod touch, and Mac. http://support.apple.com/HT204681
10. Aloul, F., Zahidi, S., Hajj, W.E.: Two Factor Authentication Using Mobile Phones. In: Proceedings on the International Conference on Computer Systems and Applications (AICCA) (2009)
11. Bosman, E., Bos, H.: Framing Signals - A Return to Portable Shellcode. In: Proceedings of the Symposium on Security and Privacy (S&P) (2014)
12. Boutin, J.I.: The evolution of webinjects (September 2014)
13. Buescher, A., Leder, F., Siebert, T.: Banksafe Information Stealer Detection Inside the Web Browser. In: Proceedings on the International Conference on Recent Advances in Intrusion Detection (RAID) (2011)
14. Caballero, J., Grier, C., Kreibich, C., Paxson, V.: Measuring Pay-per-install: The Commoditization of Malware Distribution. In: Proceedings of the USENIX Security Symposium (USENIX Sec) (2011)
15. Chytry, F.: Apps on Google Play Pose As Games and Infect Millions of Users with Adware (February 2015)
16. Dmitrienko, A., Liebchen, C., Rossow, C., Sadeghi, A.R.: On the (In)Security of Mobile Two-Factor Authentication. In: Proceedings of the International Conference on Financial Cryptography and Data Security (2014)
17. eMarketer: Smartphone Users Worldwide Will Total 1.75 Billion in 2014 (January 2014)

18. Evers, J.: Virus makes leap from PC to PDA (Feburary 2006)
19. Exact Target: 2014 Mobile Behavior Report (February 2014)
20. Federal Financial Institutions Examination Council: Authentication in an Internet Banking Environment (2005)
21. Gühring, P.: Concepts against Man-in-the-Browser Attacks (September 2006)
22. inazaruk: "Activating" Android applications (December 2011)
23. Kawamoto, D.: Cell phone virus tries leaping to PCs (September 2005)
24. Kharouni, L.: Automating Online Banking Fraud (2012)
25. Krishnan, R., Kumar, R.: Securing User Input as a Defense Against MitB. In: Proceedings of the International Conference on Interdisciplinary Advances in Applied Computing (ICONIAAC) (2014)
26. Lockheimer, H.: Android and Security (February 2012)
27. Mulliner, C., Borgaonkar, R., Stewin, P., Seifert, J.P.: SMS-Based One-Time Passwords: Attacks and Defense. In: Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA) (2013)
28. Neugschwandtner, M., Lindorfer, M., Platzer, C.: A View to a Kill: WebView Exploitation. In: Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET) (2013)
29. Oberheide, J., Miller, C.: Dissecting the Android Bouncer (Jun 2012)
30. Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., Vigna, G.: Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2014)
31. Rafael Fedler, M.K., Schutte, J.: An Antivirus API for Android Malware Recognition. In: Proceedings of Malicious and Unwanted Software: "The Americas" (MALWARE), 2013 8th International Conference (2013)
32. Sams, B.: Microsoft confirms Edge will sync passwords, bookmarks, tabs, and more. `http://www.neowin.net/news/microsoft-confirms-edge-will-sync-passwords-bookmarks-tabs-and-more`
33. Schartner, P., Bürger, S.: Attacking mTAN-Applications like e-Banking and mobile Signatures. Tech. rep., Univeristy of Klagenfurt (2011)
34. Sood, A.K., Enbody, R.J., Bansal, R.: The art of stealing banking information — form grabbing on fire (November 2011)
35. Statista: Global smartphone sales to end users 2007–2014 (2015)
36. Wang, T., Lu, K., Lu, L., Chung, S., Lee, W.: Jekyll on iOS: When Benign Apps Become Evil. In: Proceedings of the USENIX Security Symposium (USENIX Sec) (2013)
37. Wang, Z., Stavrou, A.: Exploiting Smart-Phone USB Connectivity For Fun And Profit. In: Proceedings of the Computer Security Applications Conference (ACSAC) (2010)
38. Wyke, J.: What is zeus? Sophos (May 2011)